

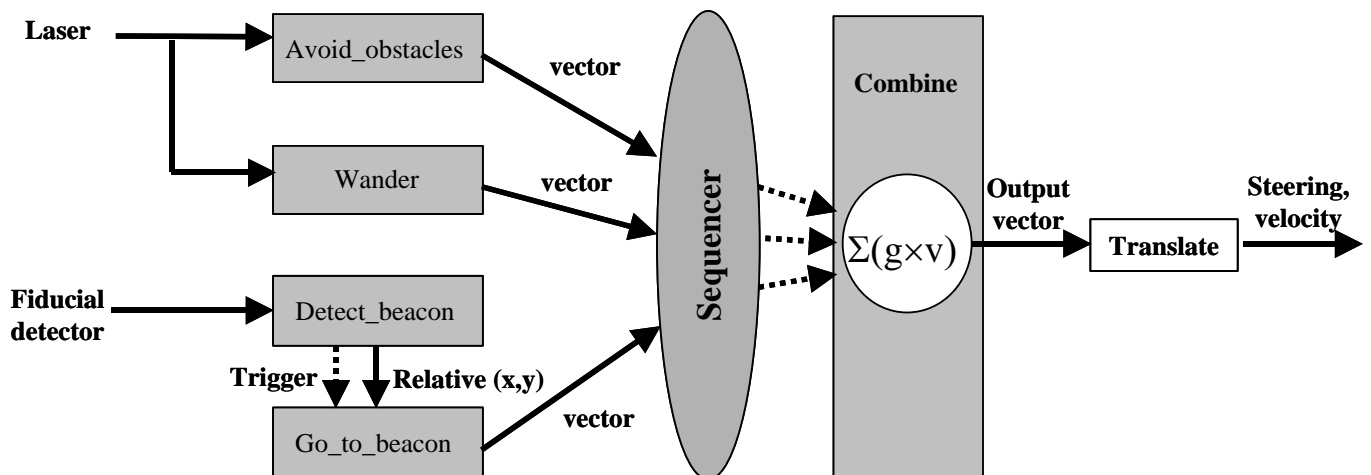
Homework 3: Combining Multiple Behaviors

Assigned: Thursday, Feb. 8, 2007
Due: Friday, Mar. 2, 2007 at 23:59:59

As we have discussed, behaviors for behavior-based autonomous robots are usually built up incrementally. This assignment will generate robot control code with multiple behaviors. The resulting robot will have 3 behaviors: avoid_obstacles, wander, and go_to_beacon, as well as a triggering function called detect_beacon. The robot control will be based upon motor schemas, with outputs of individual behaviors being vectors that are cooperatively summed to generate the resultant output control vector for the robot. The behaviors whose outputs contribute to the summed vector at each point in time are determined by a sequencer that determines which behaviors are appropriate during each part of the task. In addition, the go_to_beacon behavior will only be activated after the detect_beacon has actually found a beacon (thus triggering the go_to_beacon behavior).

This assignment guides you through a step-by-step development of this robot control code. You don't necessarily have to develop the code in exactly the specified order. If you prefer to develop it in some other order, obviously you can. The final robot behavior should cause the robot to wander around avoiding obstacles until it detects a previously unvisited beacon (which is a lot like a "waypoint" in your Homework #2), then going to the location of the beacon using the go-to-beacon behavior. Once the beacon is reached, the Sequencer should cause the robot control code to go back into wander mode to search for another (different) beacon. This repeats infinitely. The following diagram gives the overall architecture of this robot control code.

The results you are to turn in for this assignment are listed at the end of this document.



0.a. Simulation of beacon. In this exercise, a beacon will be detected as a fiducial. (Note: a *fiducial* is a special marker that can be easily detected with some appropriate sensor. For example, the fiducial could be a reflective barcode that can be detected easily by a laser. Or, it could be a distinctive visual feature that is easily detected by a camera. For this exercise, we don't really care exactly what the fiducial looks like, or what sensor is used to detect it. We'll just use the capabilities provided for this purpose in Player/Stage). To create a beacon, you'll need to define it in your world model. Here is the definition you should use (put this in your world file):

```
define beacon model(  
  size [ 0.3 0.3 ]  
  gui_movemask 3  
  gui_nose 0  
  fiducial_return 10  
)
```

Now, we'll define 3 specific beacons (also put this in your world file):

```
beacon( pose [0 0 0 ] color "red" )  
beacon( pose [5 5 0 ] color "purple" )  
beacon( pose [8 -5 0 ] color "orange" )
```

(FYI: Here's the online documentation for the world file, which includes comments on parameters specific to the fiducials: http://playerstage.sourceforge.net/doc/Stage-2.0.1/group__model.html, so that you can understand what the various parameters mean).

0.b. Setting up everything. In this exercise, you must use the new bitmap called "oneRoomWalls.png", provided on the course website (under Assignment #3, here:

<http://www.cs.utk.edu/~parker/Courses/CS594-spring07/Assignments/oneRoomWalls.png>).

Define your window and map (in your world file) as follows:

```
window(  
  size [ 755.000 565.000 ]  
  center [0 0]  
  scale 0.04  
)  
  
map(  
  bitmap "bitmaps/oneRoomWalls.png"  
  map_resolution 0.02  
  size [30 20]  
  name "oneRoomWalls"  
)
```

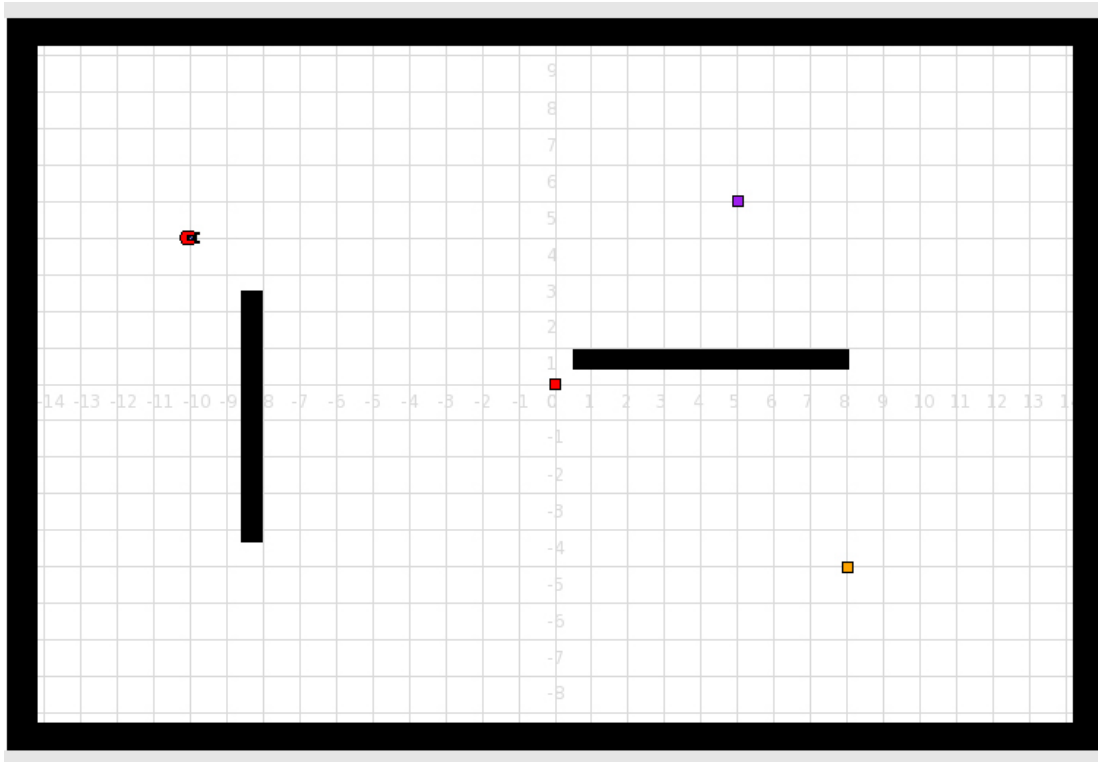
In this exercise, your pioneer2dx robot will make use of a laser scanner and a fiducialfinder (which is a sensor to find fiducials ☺). Be sure you're still using gps for robot localization. Your robot's starting pose must be [-10 4 0]. Here is a robot definition for your world file that has the appropriate parameter settings:

```
define HW3pioneer pioneer2dx  
( sick_laser (  
  pose [0.030 0.000 0.000 ]  
  fiducialfinder( range_max 8 range_max_id 5 )  
)  
  
  fiducial_return 1  
  localization "gps"  
  localization_origin [ 0 0 0 ]  
)  
  
HW3pioneer  
( name "robot1"  
  pose [-10 4 0]  
  gripper( pose [0.200 0.000 0.000] color "gray" )  
)
```

(FYI: Here's more relevant online documentation:

http://playerstage.sourceforge.net/doc/Stage-2.0.1/group__model__fiducial.html)

If you've done everything correctly, your starting Stage view should look like this:



1. Avoid_obstacles. Write an “avoid_obstacles” robot behavior that uses the laser scanner to avoid obstacles. The perceptual schema for this behavior is the detection of a nearby obstacle in the direction of motion of the robot. The motor schema for this behavior is the generation of an output vector that turns the robot away from the obstacle. Note that instead of generating “SetSpeed” commands, your code now generates an output vector giving the desired direction and magnitude of motion.

2. Wander. Write a “wander” robot behavior. The perceptual schema for this behavior is a potential field that occasionally changes the robot motion by some random direction and velocity amount, within pre-specified turning and velocity ranges. If you want, you can make use of the laser to give preference to certain directions when generating new directions (but only if you want to; this isn't necessary.) Make use of time of day as a seed to your random number generator, so that your robot exhibits different behavior on different runs. The motor schema for this behavior is the generation of an output vector that turns the robot in the specified random way.

3. Combination of output vectors. Write a function called “vector_combine” that accepts output vectors from multiple behaviors and generates a summed output vector by multiplying the individual behavior vectors by a gain matrix (G), and then using vector addition. It is up to you to define the gain matrix so that your robot behaves well.

4. Conversion of output vector to motion commands. Write code that accepts a resultant output vector from the behavior combination function and converts it to a “SetSpeed” command to control the robot's motion.

5. Test wander + avoid_obstacles + vector combination + output vector conversion. Test your code from parts 1 through 4 to ensure that your robot successfully wanders about its environment avoiding obstacles.

6. Detect_beacon. Write a “detect_beacon” function. This function uses the robot’s fiducialdetector to detect a nearby beacon that has not previously been visited. Here, we’ll assume a beacon has been visited if the robot gets within a distance of 1 meter from the beacon. Note that this means the robot keeps a memory of beacons it has seen before (e.g., based on the beacon’s position). The first time a new (i.e., unvisited) beacon is seen, this function triggers the “go_to_beacon” behavior, outputting the relative (x,y) position (i.e., relative to the robot) of the beacon to go to. Subsequent to the first detection, this function continues to output the current relative (x, y) position of that beacon, thus serving as the perceptual schema for the go_to_beacon behavior.

This function will make use of the FiducialProxy; online documentation is here:

http://playerstage.sourceforge.net/doc/Player-2.0.0/player/classPlayerCc_1_1FiducialProxy.html

One thing to be aware of: the fiducialdetector can detect more than 1 fiducial at a time, stored in an array. You should be sure your code can handle multiple beacons within the robot’s field of view, and that the beacon selected by this function does not oscillate between different visible beacons. That is, as soon as a robot detects a new beacon, it should “fix” on that beacon until it is visited. For this assignment, your code SHOULD NOT remember the location of other detected (but not “fixed upon”) beacons, with the idea that you can use this information to help guide the robot more efficiently to the next beacon. While that would potentially make your robot smarter, we’ll deal with planning issues another time. In summary, beacons can only be discovered through wandering, and the robot can only focus on one beacon at a time.

7. Go_to_beacon. Re-use parts of your Assignment #2 code for navigating to a goal to generate a behavior for go_to_beacon. Here, the beacon position returned by the “detect_beacon” function is the robot’s perceptual schema (acting like a waypoint from Assignment #2), and the go_to_beacon motor schema generates an output vector that turns the robot toward the beacon. Again, the robot’s visitation of a beacon ends when it reaches within 1 meter of that beacon.

8. Test detect_beacon and go_to_beacon. Test your code from parts 6 and 7 to ensure that the robot can detect a beacon and go to it.

9. Add Sequencer module. To your debugged code from parts 5 and 8, add a Sequencer module that decides when the outputs of each behavior are relevant for the current situation, only allowing the relevant behaviors to contribute to the summed output vector. [This module will be the bulk of your “main” function in your code.] At different times, different behaviors will be in use, as follows:

- Initially, combine the outputs from “wander” and “avoid_obstacles”, with the “detect_beacon” function also active to see when it is time to trigger the “go_to_beacon” behavior.
- When “detect_beacon” has found the beacon, ignore the output from “wander”, and include output from “go_to_beacon” (as well as avoid_obstacles, which will always be active, except perhaps when approaching a beacon).
- When the beacon position is reached, the “detect_beacon” function should note the position of the beacon, so that it won’t be visited again.
- Repeat above infinitely.

10. Test complete system. Test your complete system using the setup described earlier. Your resulting code should cause the robot to wander around, avoiding obstacles, and visiting previously unvisited beacons in the environment. If you’ve written a good wander function, and you let the robot run long enough, it should eventually visit all the beacons in the environment. (Of course, your robot won’t know it’s done, so it will keep looking, even after all the beacons have been visited.)

WRITE UP THE FOLLOWING (written up in a single pdf file called *yourlastname-HW3.pdf*):

- a) The definition of the specific mathematical functions you used to generate the output vectors for avoid obstacles, wander, and go_to_beacon, including a definition of the magnitude and direction of each output vector.
- b) The definition of the specific gain matrix you used to weight the output vectors from the various behaviors.
- c) The definition of the specific mathematical function you used to convert the resultant output vector to a robot “SetSpeed” command.
- d) 3 screenshots of your program in operation (from 3 separate runs) that illustrate your program’s ability to find beacons while wandering and avoiding obstacles. Be sure to turn on the robot trace. Ideally, these runs will show the robot finding at least 2 of the beacons in a single run. It is even better if you show your robot finding all 3 beacons in a single run. [These runs should be different from each other, due to the randomness of the wander behavior.]

SUBMITTING YOUR HOMEWORK:

Place all your files in a single directory. These files should include:

- Your written answers to the questions above, plus screen dumps as requested (i.e., the file “yourlastname-HW3.pdf”)
- Your configuration file, called “HW3.cfg”
- Your world file, called “HW3.world”
- Your makefile, called “makefile” or “Makefile”
- Your robot control code, called “yourlastname-HW3.cc” .
- Any additional include files you created (called whatever you want them to be called).

Remove all other unnecessary files.

Use the submit script **594sir_submit** to submit your files. (These will be emailed to Dr. Parker.)

[When I unpack your files, I should be able to say “make yourlastname-HW3”, and then “player HW3.cfg”, and then “./yourlastname” to run your code. Be sure you have everything set up properly so that this works as expected.]